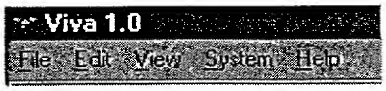
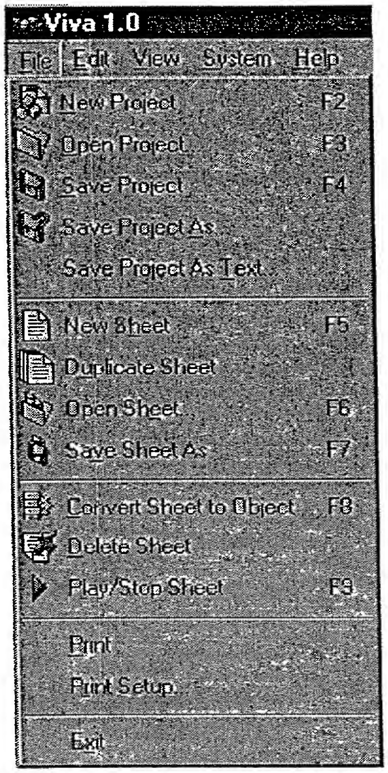
Menu Commands



The Menu Commands of VIVA allow you to display, enter, edit, and save VIVA programs.

File Menu	Load and display VIVA projects
Edit Menu	Graphical editing, cutting and pasting
View Menu	View Options
System Menu	Target system selection and control
Help Menu	Assistance to the user.

File Menu

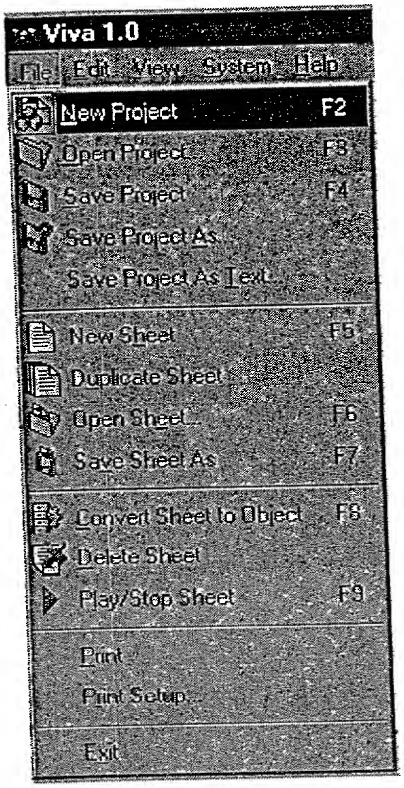


The file menu deals with loading, saving, and executing VIVA programs. Many of these commands are available as Function Keys, and from the ToolBar by clicking on the corresponding icon.

New Project	Clears all objects, pages, and Modules.
Open Project	Loads and displays a VIVA project.
Save Project	Saves the current project.
Save Project As	Saves the current project with the name you specify.
New Sheet	Creates a new sheet or Behavior Page.
Duplicate Sheet	Saves and renames the current project.
Open Sheet	Loads a sheet from a file.
Save Sheet As	Saves the current sheet with the name you specify.
Convert Sheet to Object	Captures sheet behavior as a VIVA Module.
Delete Sheet	Erases and deletes the current Behavior Page.
Play/Stop	Executes the behavior on the displayed Behavior Page.
Print	Prints the current sheet.
Print Setup	Sets up the printer options you specify.

Exit Quits VIVA.

New Project Command



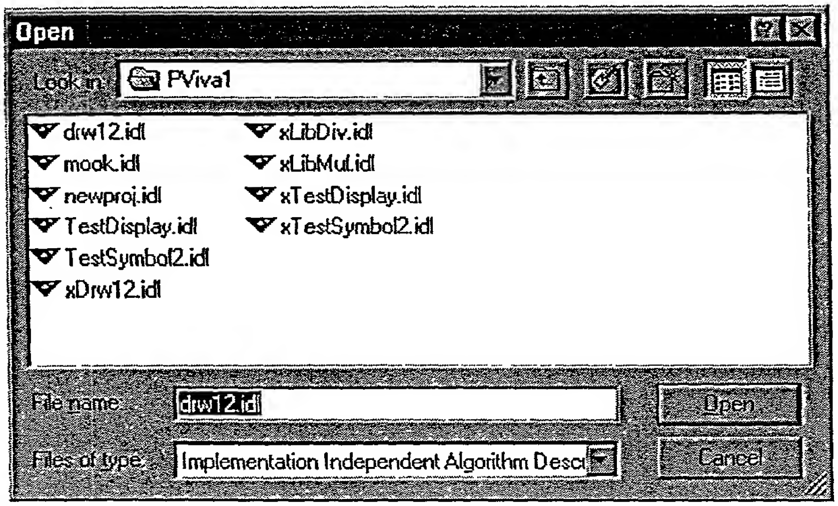
Clears the current Project and Behavior Pages. This function is equivalent to a reset of VIVA to the initial conditions.

WARNING

Be sure to save your work to a file before selecting this function. The New Project function deletes all work. **All work may be lost.**

Open Project Command

Retrieves a Saved Project from disk.



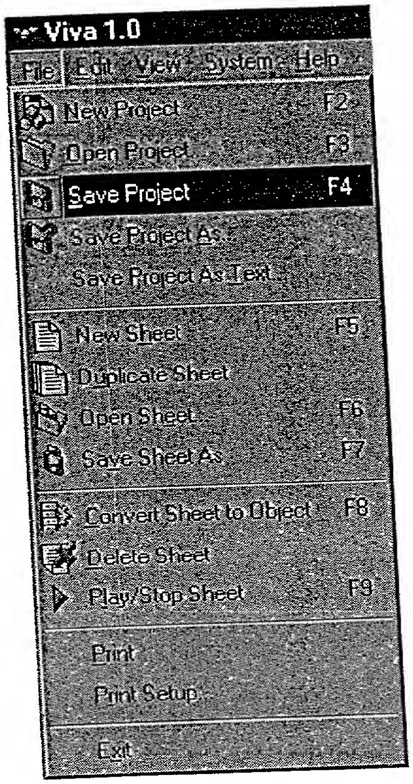
Look in: PViva1

drw12.idl
mook.idl
newproj.idl
TestDisplay.idl
TestSymbol2.idl
xDrw12.idl
xLibDiv.idl
xLibMul.idl
xTestDisplay.idl
xTestSymbol2.idl

File name: drw12.idl

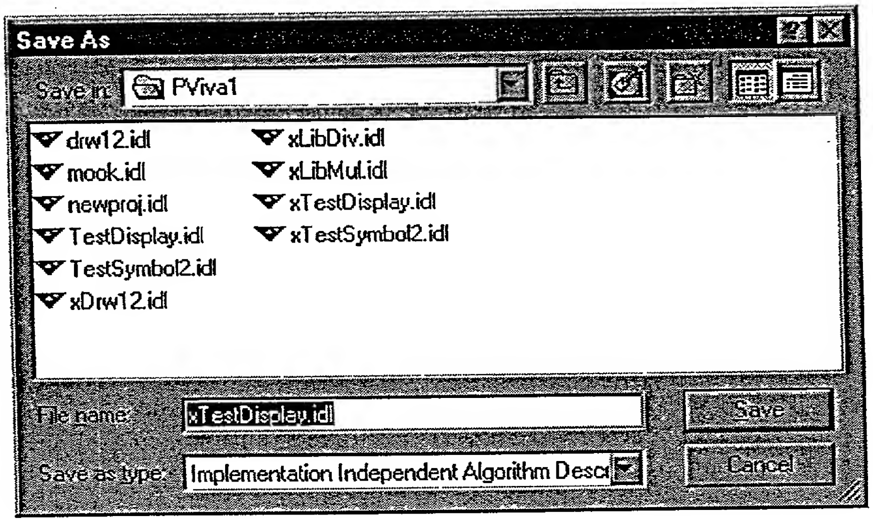
Files of type: Implementation Independent Algorithm Desc

Save Project Command



This saves an entire project. In contrast to Save Sheet As, this procedure saves all of the sheets (Behavior Pages), all of the data sets, the UI Form, and everything else associated with a project. Because of the more extensive nature of SaveProject, a different file extension is used. After a project has been saved, it can be retrieved using Open Project

Save Project As Command



Save As

Save in: PViva1

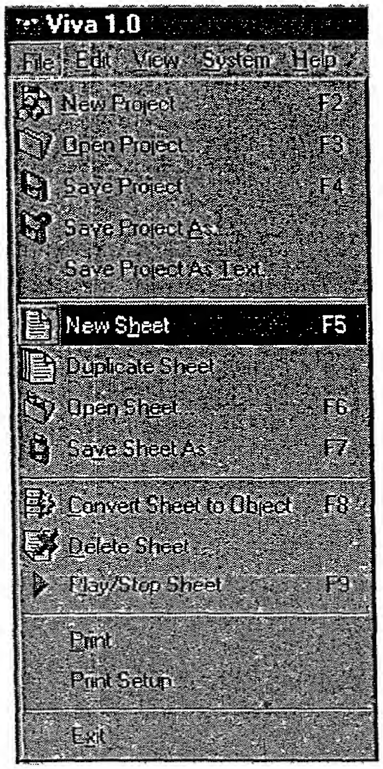
drw12.idl
mook.idl
newproj.idl
TestDisplay.idl
TestSymbol2.idl
xDrw12.idl
xLibDiv.idl
xLibMul.idl
xTestDisplay.idl
xTestSymbol2.idl

File name: xTestDisplay.idl

Save as type: Implementation Independent Algorithm Desc

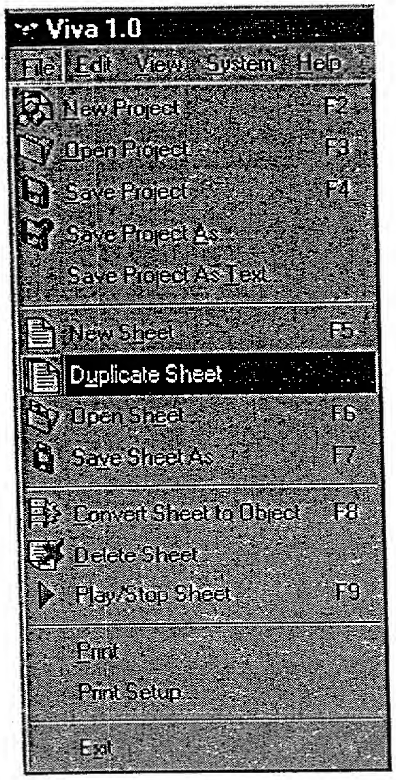
This saves an entire project with the new name you provide. As with Save Project, this procedure saves all of the sheets (Behavior Pages), all of the data sets, the UI Form, and everything else associated with a project. After a project has been saved, it can be retrieved using Open Project

New Sheet Command 🗎



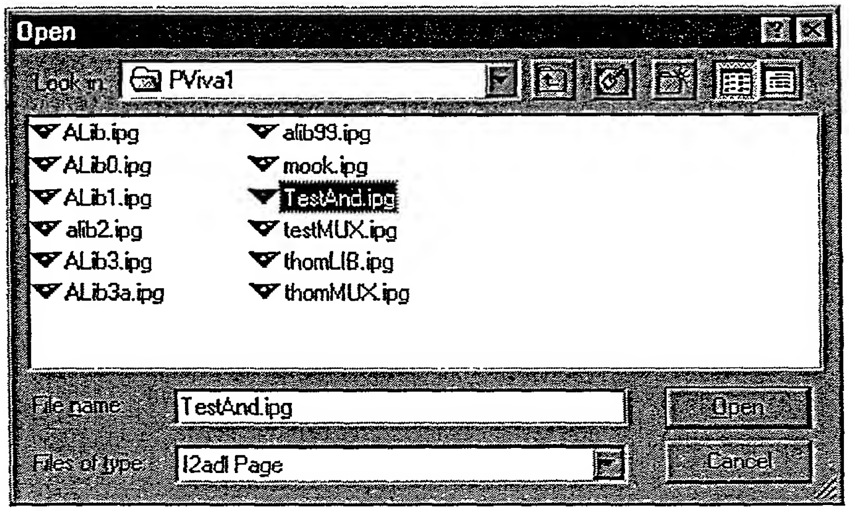
Use this command to create a new Behavior Page. The old Behavior Page is not lost, but can be reactivated by clicking on it in the Behavior Page Tree.

Duplicate Sheet Command



Use this command to duplicate the current Behavior Page.

Open Sheet Command



Open

Look in: PViva1

ALib.ipg
ALib0.ipg
ALib1.ipg
alib2.ipg
ALib3.ipg
ALib3a.ipg
alib99.ipg
mook.ipg
TestAnd.ipg
testMUX.ipg
thomLIB.ipg
thomMUX.ipg

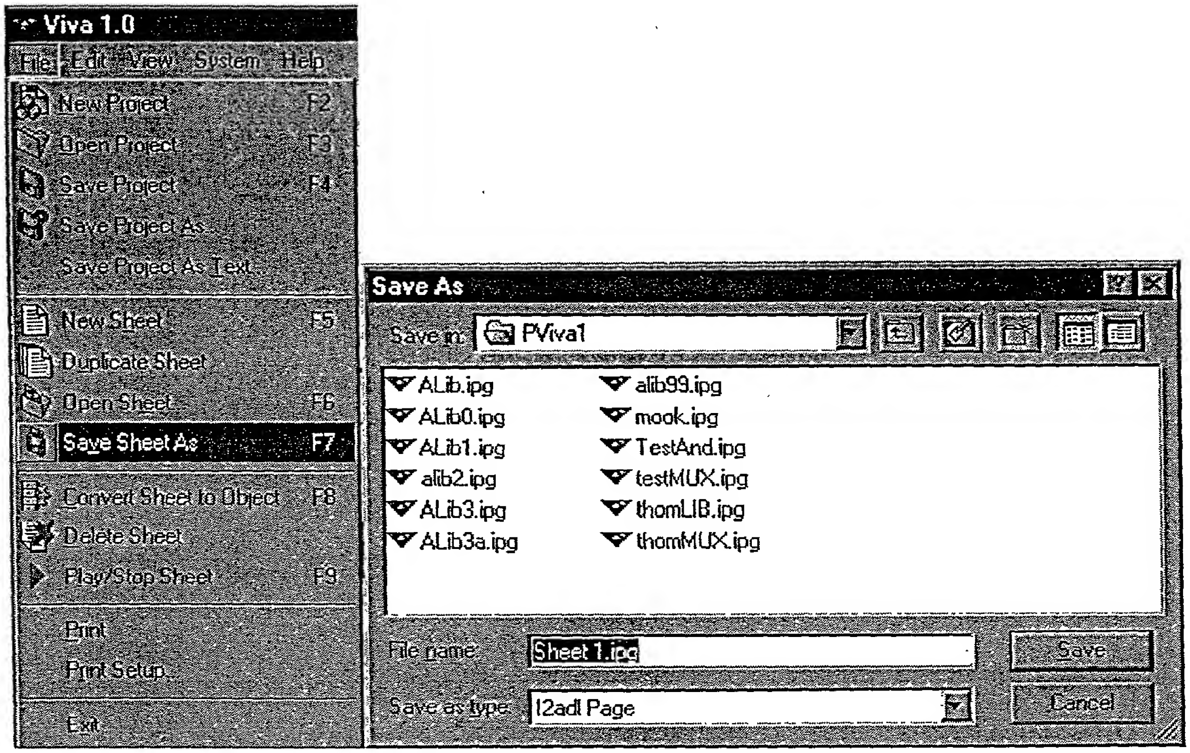
File name: TestAnd.ipg

Files of type: I2adl Page

This loads a Behavior Page from a file.

Save Sheet As Command

This saves a Behavior Page to a file with the name you specify. Because there is no limit to the number of VIVA Modules you can place on a Behavior Page, this feature enables you to create Librariesof VIVA Modules.

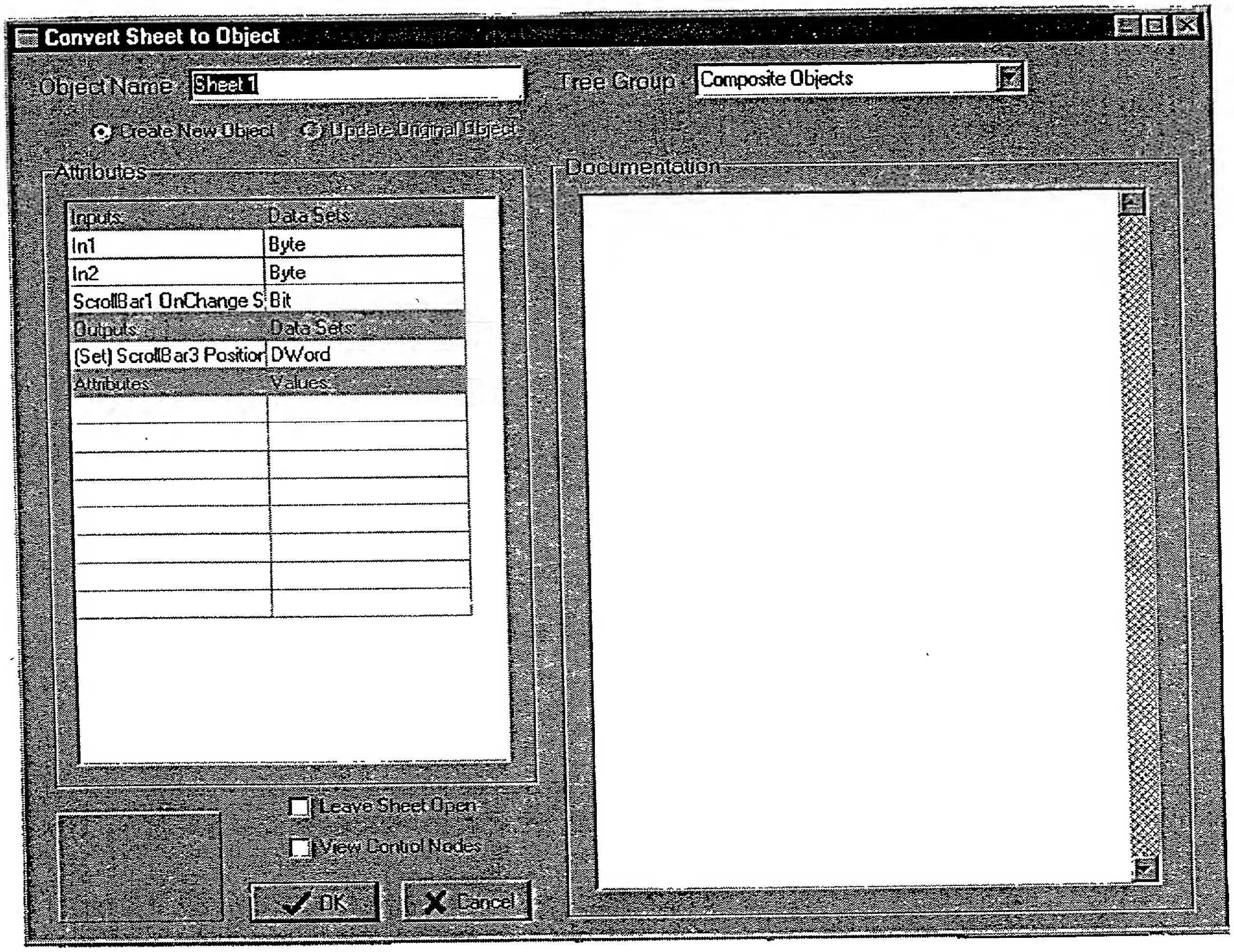


File name: Sheet1.ipg

Save as type: I2adl Page

Convert Sheet To Object Command

Once you have defined a Behavior Page for a VIVA Module, you can then convert the Behavior Page into a VIVA Module. Inputs and outputs of the Module are the same order and type as defined for the Behavior Page. Before saving the Module, you are given a final chance to enter the information about the Module (things like giving the Module a name, naming the input and output nodes, and documenting the function of the Module and how it is intended to be used). This is done through the Convert Sheet Dialog Box. To maintain universality, this interface is essentially the same as the Edit Attributes Dialog Box.



Convert Sheet to Object

Object Name: Sheet 1

Tree Group: Composite Objects

● Create New Object ○ Update Original Object

Attributes

Inputs:	Data Sets:
In1	Byte
In2	Byte
ScrollBar1 OnChange S	Bit
Outputs:	Data Sets:
(Set) ScrollBar3 Positior	DWord
Attributes:	Values:

Documentation

□ Leave Sheet Open

□ View Control Nodes

Notes:

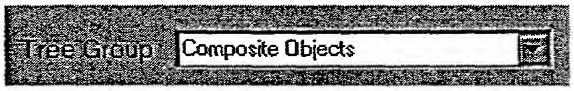
1. This dialog is probably the best interface provided for naming Module nodes. We recommend you name Module nodes here.

2. Documentation should be provided at this point, to define the function of the Module, its intended uses, and restrictions in its use.

3. Overloading is a characteristic that allows you to write several versions of the same behavior to handle the various implementation requirements. As a result, you can have three or four different addition Modules. In many of these cases, you will use a prior implementation as a template for the new

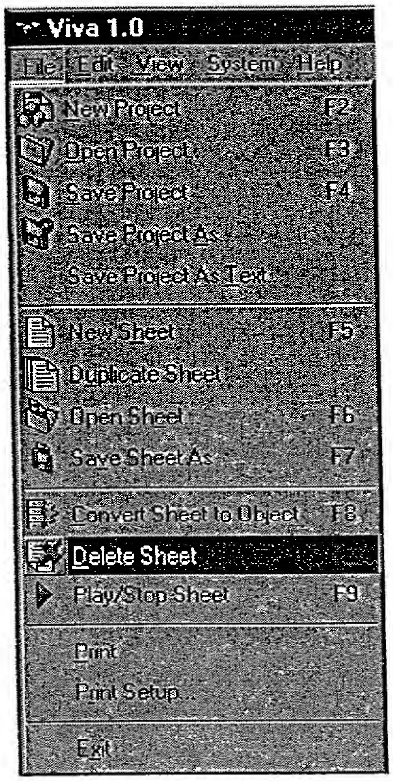
behavior. In these instances, you have the option of either updating the existing Module or creating a new one.

4. You also have the option of selecting for the Module the appropriate Tree Group in the Object Tree. Typical Tree Groups include Bit Operations, Arithmetic Operations, Test Modules, and so forth.



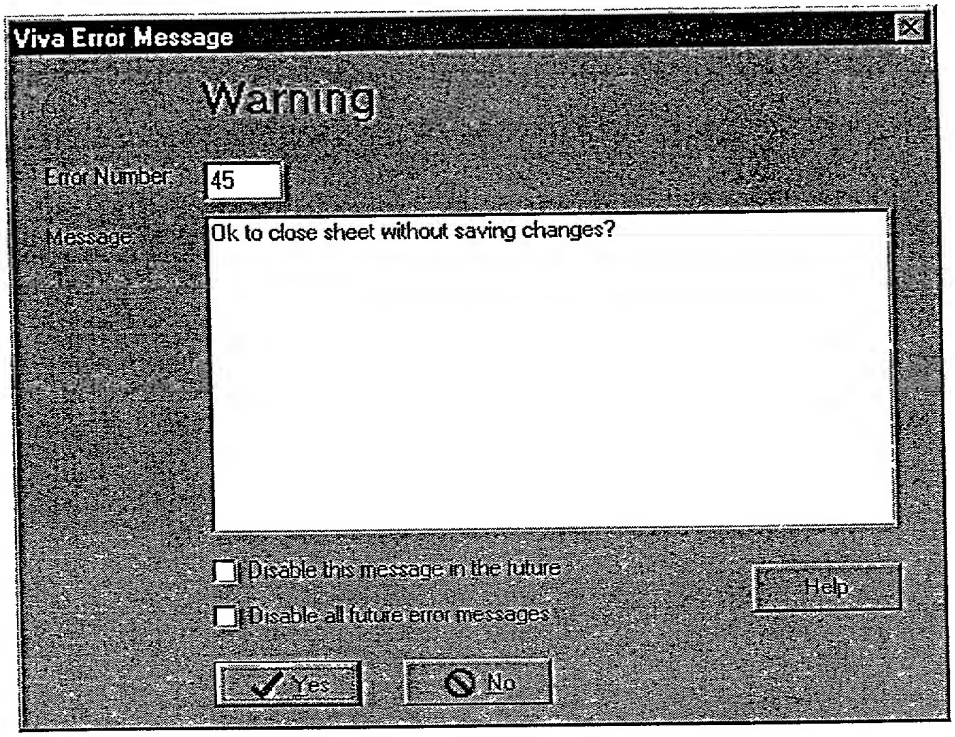
Tree Group: Composite Objects

Delete Sheet Command



This command closes the currently displayed sheet. **Any information on the sheet is lost.**

You will receive a warning screen to verify that you really do want to delete the sheet.



Viva Error Message

Warning

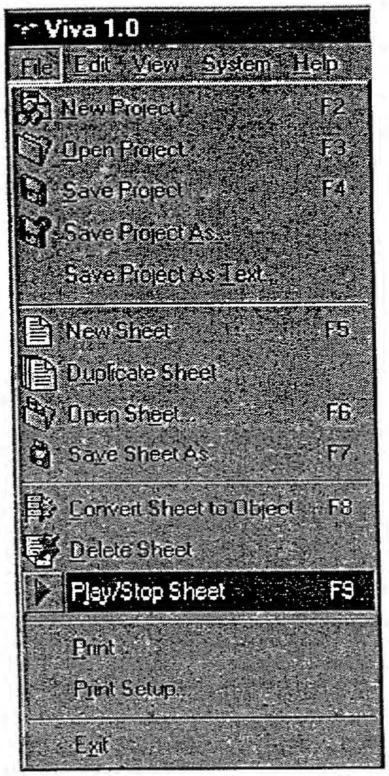
Error Number: 45

Message: Ok to close sheet without saving changes?

□ Disable this message in the future
□ Disable all future error messages
Help
✓ Yes
⊘ No

Run Behavior Page ▶

When you click on the Play/Stop button, VIVA will execute the Program specified by the current visible Behavior Page.

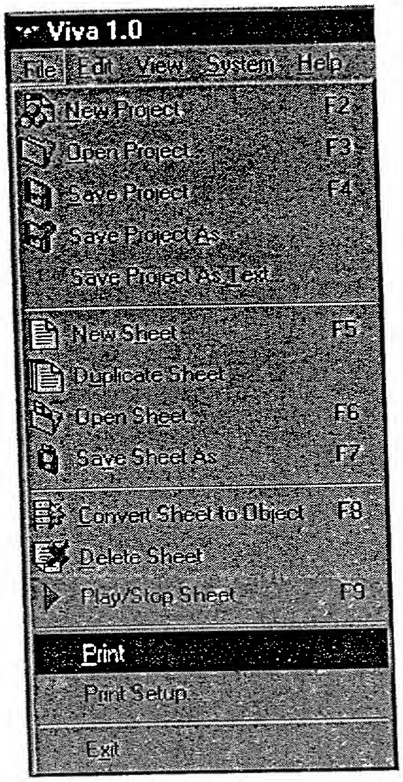


Inputs and Outputs to the process will be the Inputs and Outputs specified on the currently visible Behavior Page.

RESTRICTIONS

To execute a VIVA program, none of the Inputs on the currently visible Behavior Page can be of type Variant. Otherwise, an error will occur.

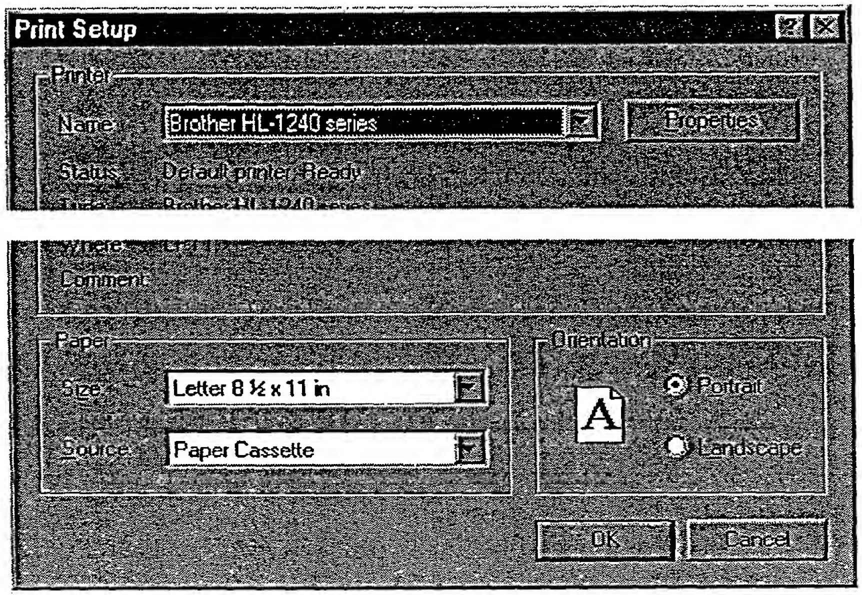
Print Command



This command prints the current sheet.

Note: This feature is not supported in the current release of VIVA.

Print Setup Command



Print Setup

Printer

Name: Brother HL-1240 series

Status: Default printer; Ready

Type: Brother HL-1240 series

Where: LPT1:

Comment:

Paper

Size: Letter 8 ½ x 11 in

Source: Paper Cassette

Orientation

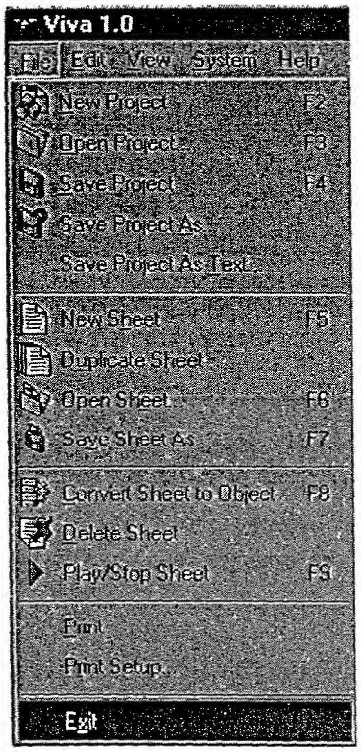
● Portrait

○ Landscape

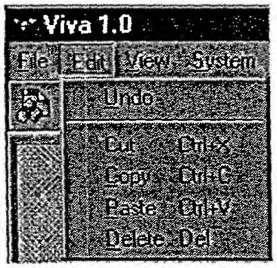
This command allows you to specify the options that will be used by the Print command.

Exit

This command closes down VIVA. **Any information you have not saved will be lost.**



Edit Menu



The Graphical User Interface of VIVA supports the Windows-standard editing Hot Keys for copy (control-C), cut (control-X), paste (control-V), and delete. The same commands can also be activated through the Edit Menu. To copy an object or group of objects, the object(s) must first be selected. This can be done in a number of ways:

1. To select a single object, click on it with the mouse.

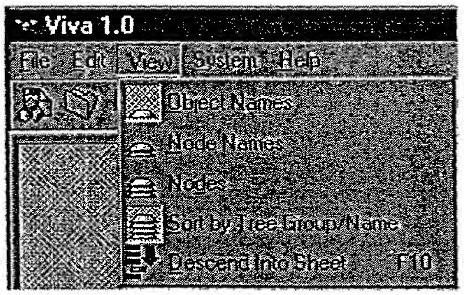
2. To select a group of objects, draw a box around the group. Click on the corner of the box. Then, while holding the left mouse button down, drag the mouse position until the box contains the desired objects.

Groups of objects can be copied between different Behavior Pages.

Note

VIVA does not use the standard Windows clipboard. As a consequence, objects cannot be moved across files, and selected objects are no longer available after VIVA has been stopped.

View Menu



The view menu allows you to control the displayed level of detail. Many of these commands are available as Hot Keys, and from the ToolBar by clicking on the corresponding icon.

View Details

View Object Names — Displays the object name above each object.

View Node Names — Displays each node name instead of each node's icon.

View Nodes — Displays node colors on Transports. (Node colors correspond to data types).

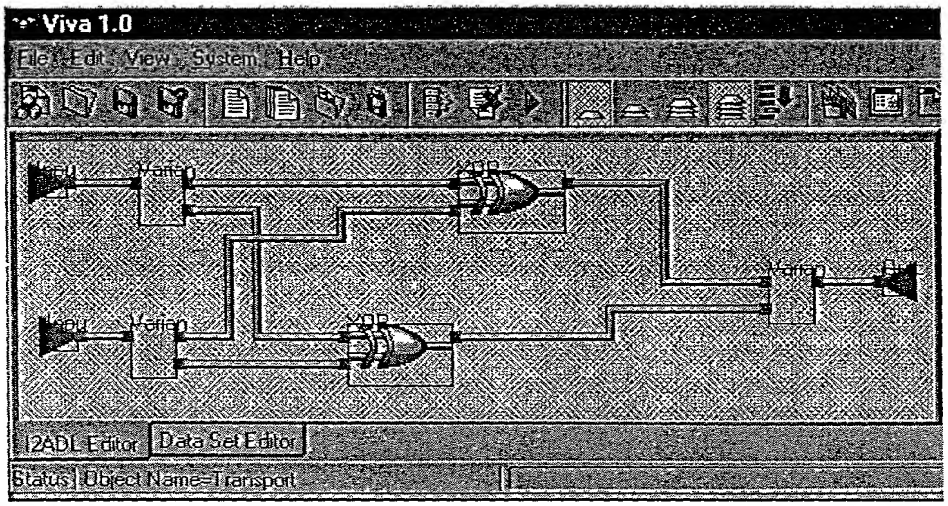
Sort by Tree Group/Name — Sorts the Object Tree in alphabetical order.

Descend into Sheet — This displays the Behavior Page of the selected object. (This feature is also available by double-clicking on the object.)

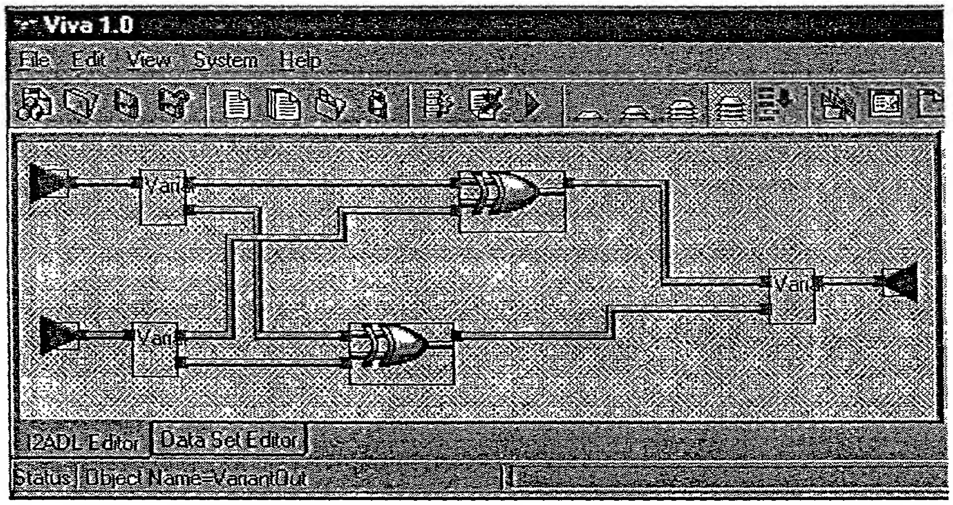
View Object Names

Allows you to view the names of the Objects or Modules on a Behavior Page. On a standard Behavior Page, the node names are suppressed.

The following graphics display a Behavior Page with and without the object names displayed.



Object Name=Transport

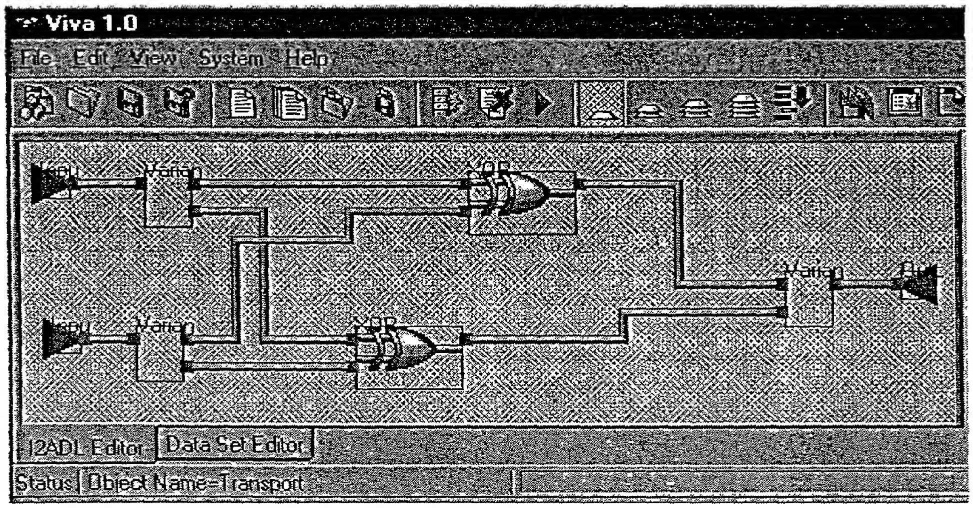


Object Name=VariantOut

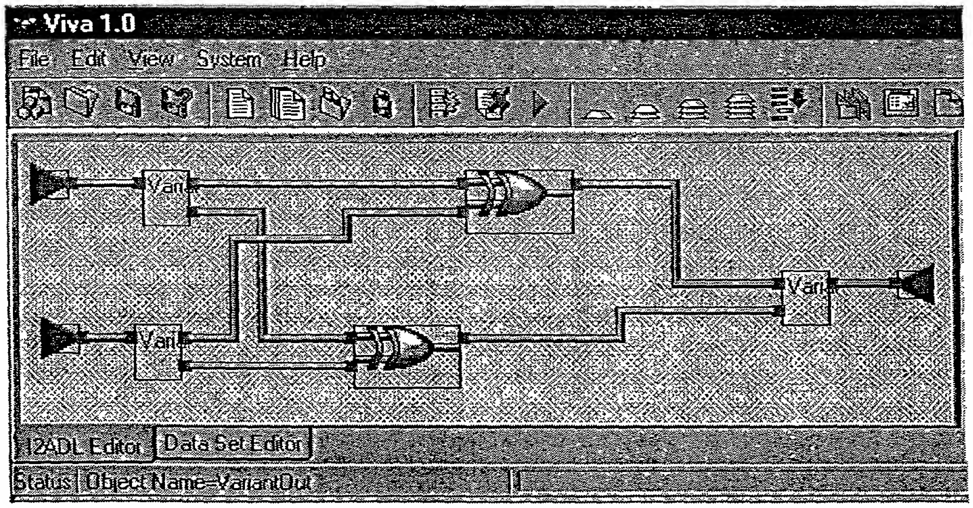
View Node Names

Allows you to view the names of the Modules or Nodes on a Behavior Page. On a standard Behavior Page, the icons of the various Modules are displayed and the names are suppressed. When Node Names are displayed, the icons are suppressed.

The following graphics display a Behavior Page with and without the Node Names displayed.



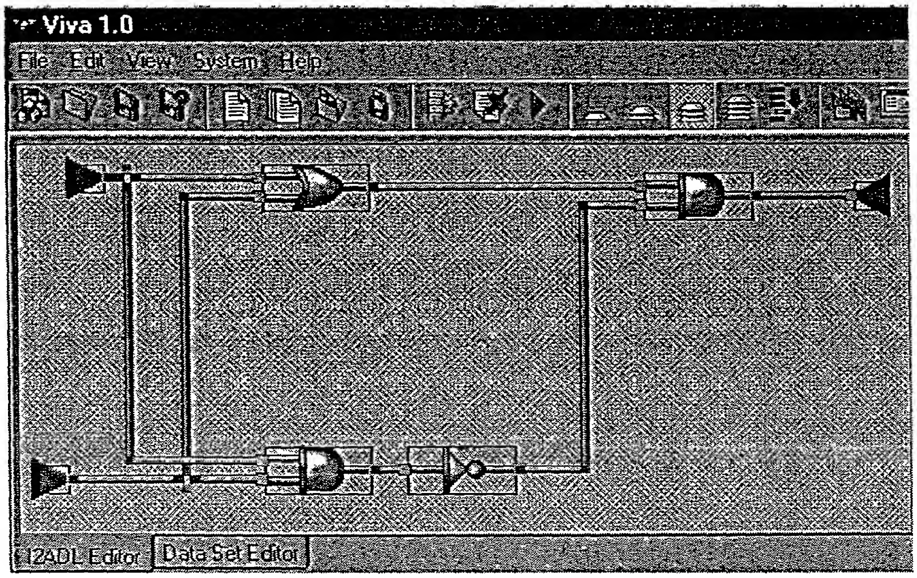
Status | Object Name=Transport



Status | Object Name=VariantOut

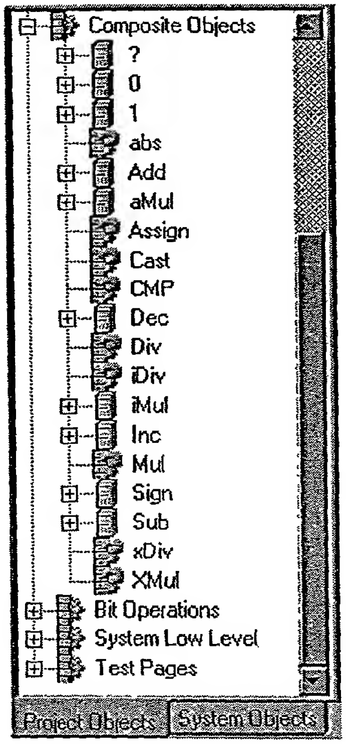
View Nodes

Node types become evident by their color if the View Node mode is activated.



Sort by Tree Group/Name

This sorts all composite objects in the Object Tree in alphabetical order.

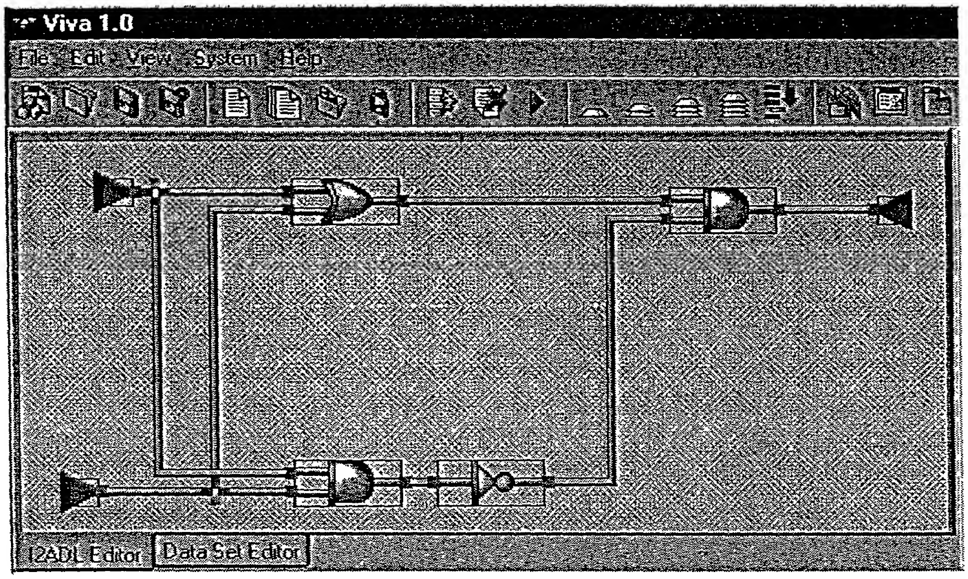


- Composite Objects
  - ?
  - 0
  - 1
  - abs
  - Add
  - aMul
  - Assign
  - Cast
  - CMP
  - Dec
  - Div
  - iDiv
  - iMul
  - Inc
  - Mul
  - Sign
  - Sub
  - xDiv
  - XMul
- Bit Operations
- System Low Level
- Test Pages

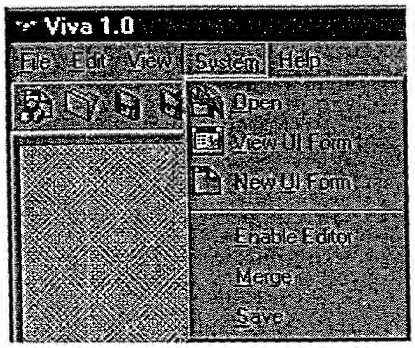
Descend Into Sheet

The Behavior Page of a VIVA Module can usually be displayed by either double clicking on the object, or by clicking on the descend icon after the Module has been selected using a left mouse click.

The following is the Behavior Page for an Exclusive OR Module.



System Menu



The System Menu enables you to select and control the Target System.

Open System Select the target system (X86 or Floating Point Gate Array).

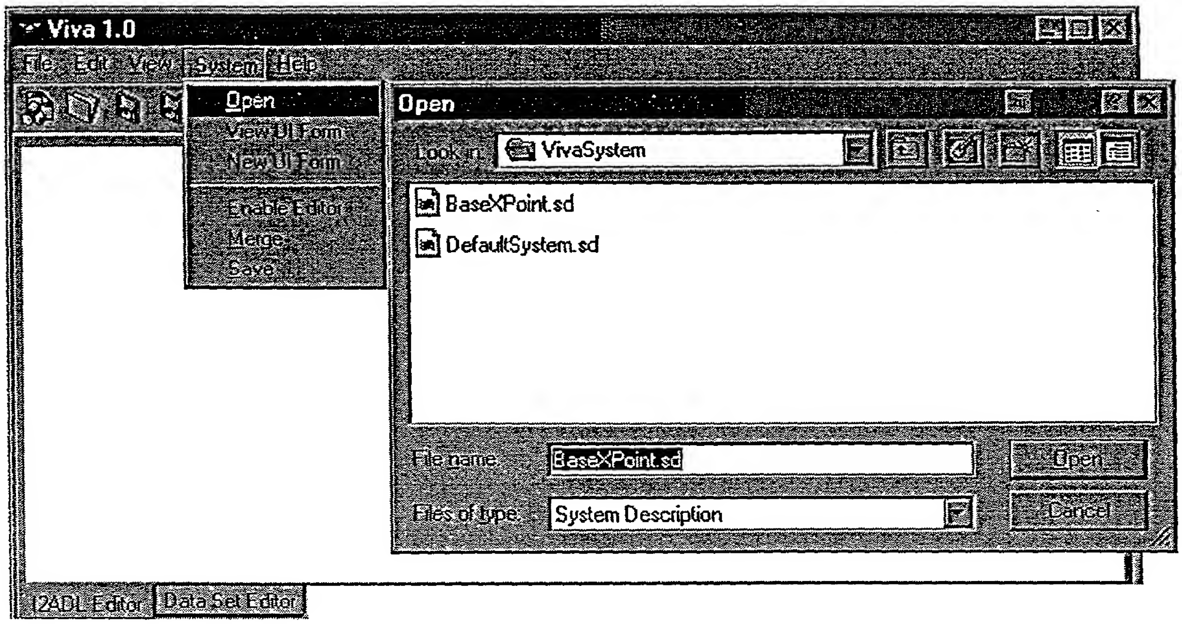
View UI Form This feature allows you to review the currently selected User Interface Form. This form controls the display and entry of data that is passed between you and VIVA.

New UI Form New UI FORM. This allows you to select a new User Interface Form. Items that appear on the form are controlled through the System Interface. In summary, any system hook or event can be placed on an interface form.

The available selections for target systems and user interfaces are all maintained in the directory C:\VIVA\VIVASYSTEM.

Merge and Save Systems are advanced features reserved for developing new system interfaces.

Open System



Look in: VivaSystem

File name: BaseXPoint.sd

Files of type: System Description

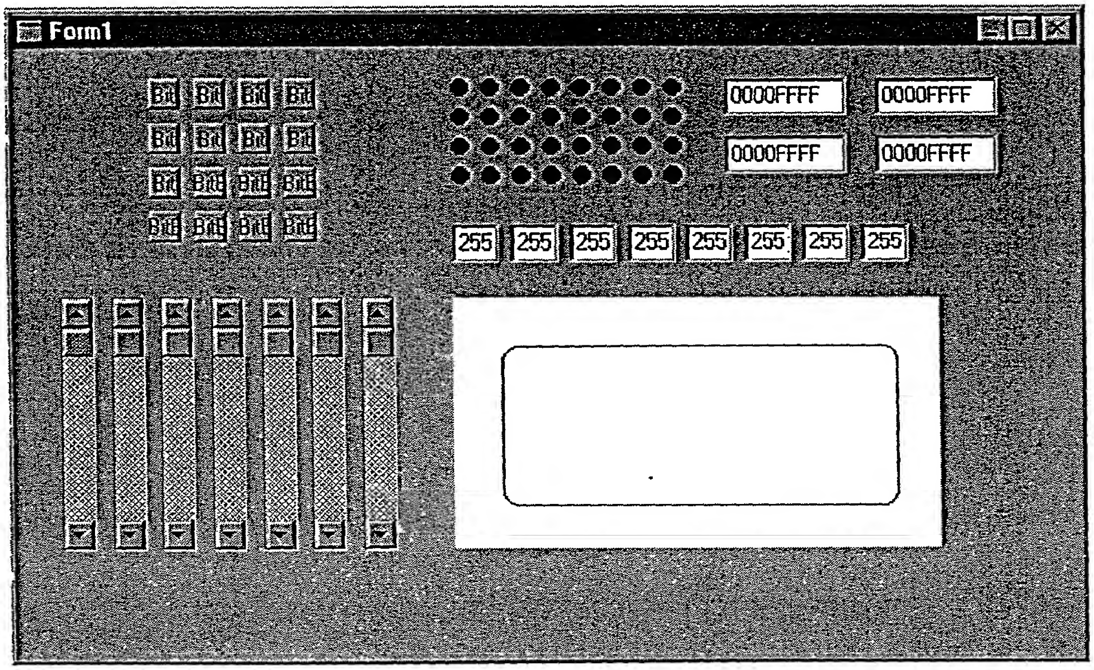
Selecting the target system. At the time of composition, the available target systems were the XILINX Floating Point Gate Array (BaseXPoint) interfaceand the X86 User Interface. To take advantage of the massively parallel capabilities of VIVA, the ultimate target would be an FPGA or other massively parallel system. The X86 system defines the interface with your computer, and allows you to develop VIVA software without requiring that an FPGA board be present.

Note: Just because the BaseXPoint FPGA interface has been selected does not necessarily mean that all computing functions will be allocated in GateWare™. In the first place, the User Interface must exist on the User's computer. External devices are allocated elsewhere. Optimization of physical allocation of resources is done by the Search Engine, and is done at compile time. The Search Engine selects from the list of available system resources in order to achieve performance at the required Information Rate.

<u>Forcing GateWare</u> To force any particular VIVA Module into GateWare, set the System Attribute String of the Module to contain the desired system.

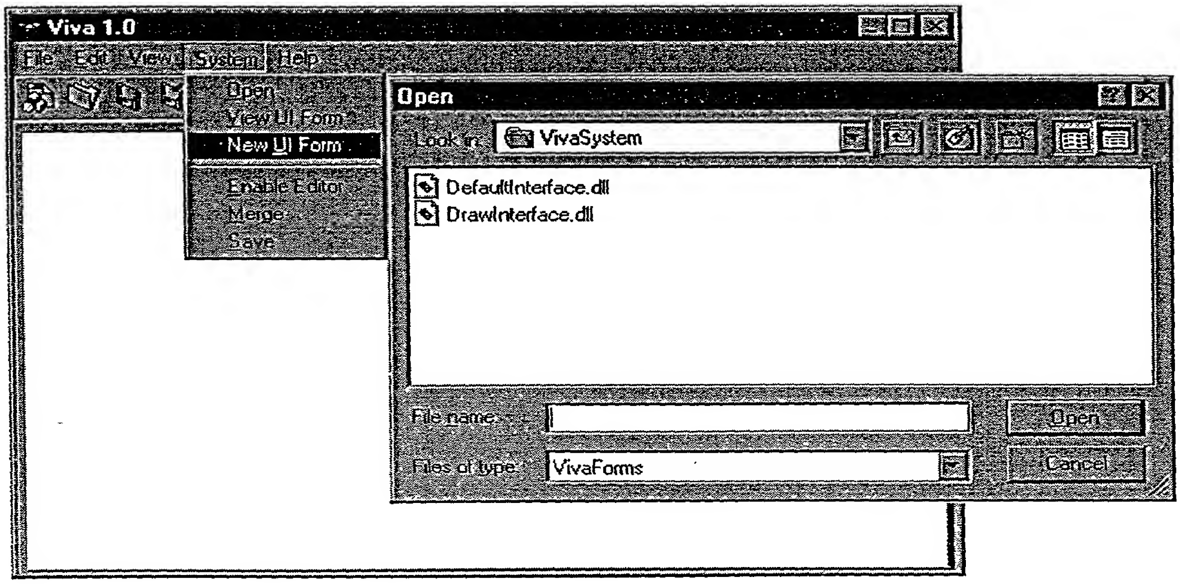
VIEW UI Form

This allows you to view the current User Interface Form, such as the one below.

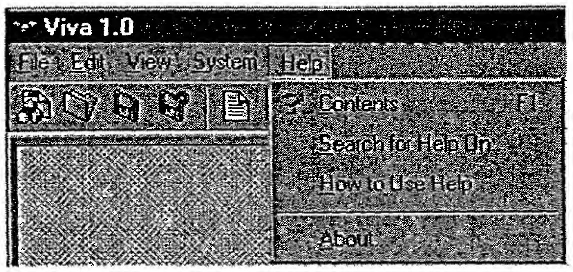


NEW UI Form

This allows you to select a new User Interface Form. Items that appear on the form are controlled through the System Interface. In summary, any system hook or event can be placed on an interface form.

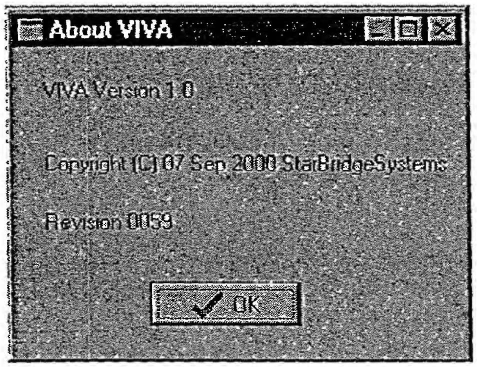


Help Menu



The Help Menu provides access to the Help System.

The About option tells the version of VIVA currently being used:



VIVA Version 1.0

Copyright (C) 07 Sep 2000 StarBridgeSystems

Revision 0059

ToolBar Controls

The Graphical User Interface of VIVA was designed to allow you to specify the desired behavior of the target computer environment.

File Commands

New Project	Clears all objects, pages, and Modules.
Open Project	Load and display a VIVA project.
Save Project	Saves the current project.
Save Project As	Saves and renames the current project projects.
New Sheet	Creates a new blank sheet.
Duplicate Sheet	Duplicates the current sheet.
Open Sheet	Loads a sheet from a file.
Save Sheet As	Saves current sheet as a file.
Convert Sheet	Captures sheet behavior as a VIVA Module.
Delete Sheet	Erases and deletes current Behavior Page.
Run/Stop	Executes the behavior on the displayed Behavior Page.

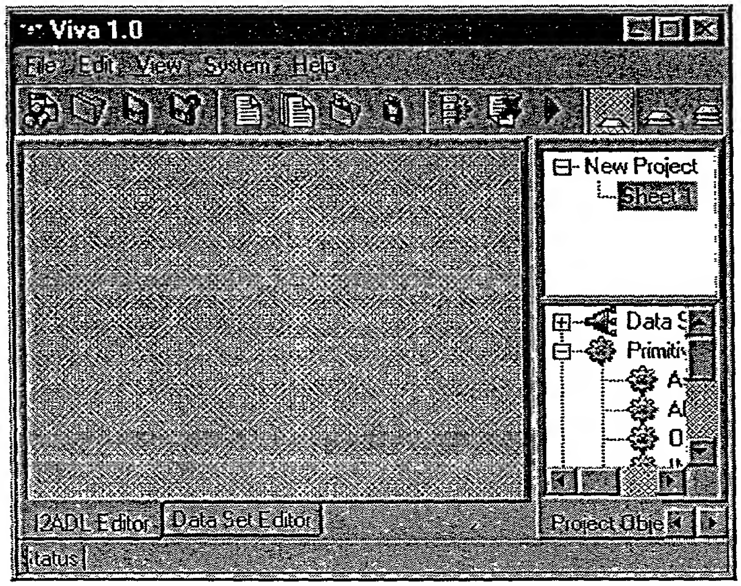
View Details

View Object Names	The names of the objects are displayed above the objects.
View Node Names	The names of the nodes of the objects are displayed instead of the object's icon.
View Nodes	The node colors are displayed on Transports. (Node colors correspond to data types.)
Sort by Tree Group/Name	Sorts the Object Tree in alphabetical order.
Descend into Sheet	Display the Behavior Page of the selected object (Also available by

	double-clicking on the object.)
Open System	Selects the target system.
View UI Form	Displays the User Interface Form.
New UI Form	Allows you to Select a new User Interface Form.

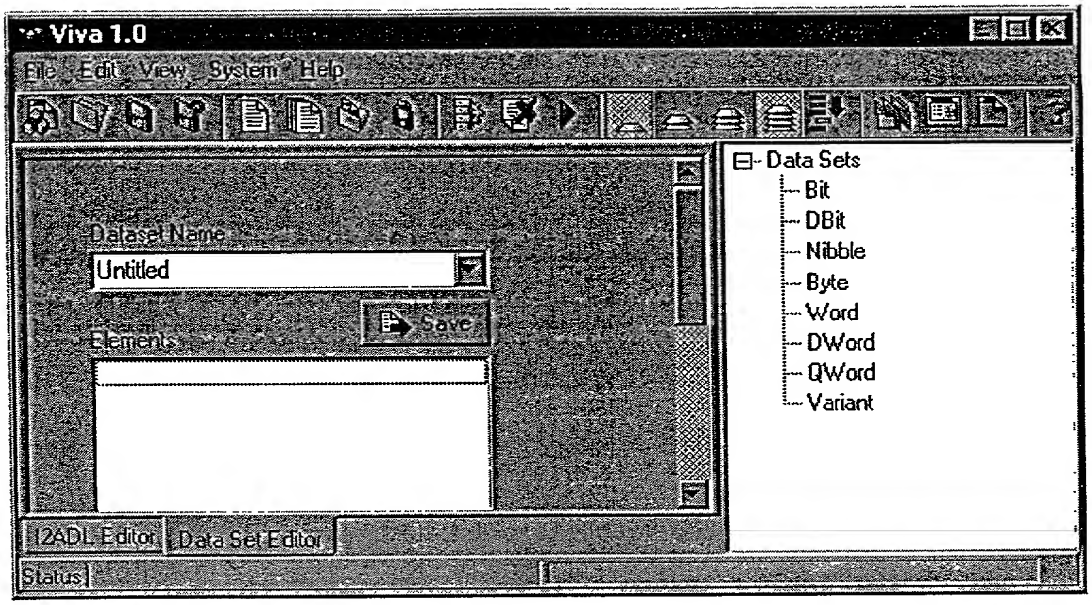
DataSet Editor

The DataSet Editor is activated by pressing the **DataSet Editor** tab at the bottom of the main window.



VIVA will then present the DataSet Editor (below). The window on the right contains the DataSet Tree, defining all of the existing data sets. To review the contents of any of these DataSets, either activate it using the drop-down menu under the DataSet Name, or drag the desired DataSet to the DataSet Name in the left pane.

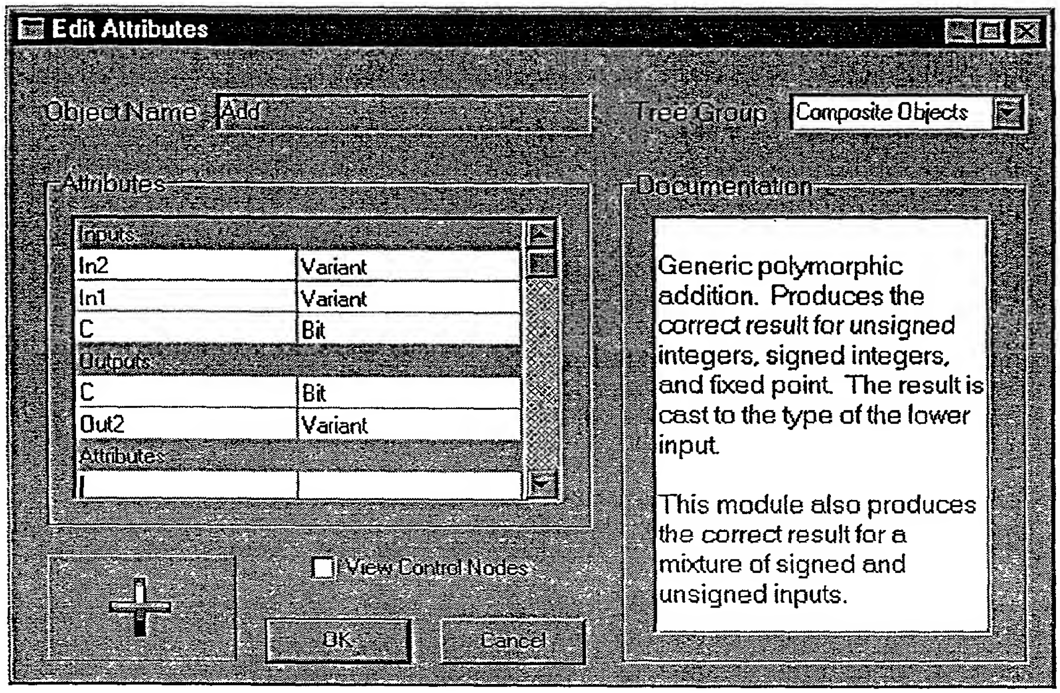
To create a new data set, first enter the new DataSet Name. Next define the components of the Data Set, by dragging them from the menu of available DataSets. (VIVA will not let you type them in as they must be preexisting and dragging eliminates the possibility of a typing error.)



When done with defining the new data set, you press the Save Button .

Note: Most of the existing software is geared toward processing DataSets with two components. Although the system allows for more, sometimes unusual results occur. Limiting the number of DataSet components to two is not really a restriction, because component parts can always be combined pair wise to make up any composite DataSet.

Edit Attributes Dialog



Object Name: Add

Tree Group: Composite Objects

Inputs:	
In2	Variant
In1	Variant
C	Bit
Outputs:	
C	Bit
Out2	Variant
Attribute:	

☐ View Control Nodes

Generic polymorphic addition. Produces the correct result for unsigned integers, signed integers, and fixed point. The result is cast to the type of the lower input.

This module also produces the correct result for a mixture of signed and unsigned inputs.

The Edit Attributes Dialog Box enables you to enter the information about a Module. The information may include giving the Module a name, naming the input and output nodes, and documenting the function of the Module and how it is intended to be used. To show the Dialog Box, right click on the Module to be affected. To maintain universality, this interface is essentially the same as the Convert Sheet to Object Dialog Box. This Dialog Box is also used for defining the DataSets for Module Inputs and Outputs.

Notes:

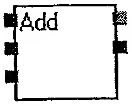
1. This dialog is can be used to name Module nodes. Node Name changes made in this dialog will apply only to the specific instance of the Module selected. The intent is to allow you to label single lines. To effect Node Name changes to all instances of the Module, it is required that the Module be selected from the Object Tree.

2. Documentation capability is provided to define the function of the Module, its intended uses, and restrictions in use. Documentation changes made in this dialog will apply to all instances of the Module.

3. You also have the option of selecting for the Module the appropriate Tree Group in the Object Tree. Typical Tree Groups include Bit Operations, Arithmetic Operations, Test Modules, etc.

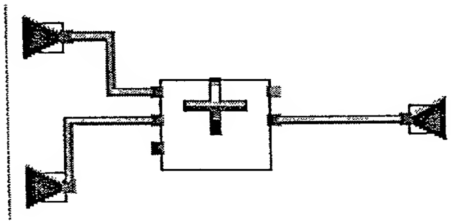
Constructing VIVA Modules

A VIVA Object or Module consists of a graphical device with an associated behavior when connected to inputs and/or outputs. When represented graphically, the input nodes are located on the left, and the output nodes on the right. For example, consider the following Add module:



This module has three inputs and two outputs. The inputs consist of two operands and an optional carry bit. The outputs consist of the sum and a carry bit. The carry bit is important for specifying the recursive definition of addition, and is useful in many applications.

To use the addition module, place it on a Behavior Page and connect it to input and output signal lines or transports. The following example accepts two inputs, adds them together, and displays the results:



To create this example first load a library or Sheet containing the addition module using the Open Sheet command 📂.

Then place two inputs and an output on the current Behavior Page by dragging them from the Object Tree . The Object Tree is located in the lower right pane of the VIVA Graphical User Interface. It contains the list of available modules.

To test addition, the types of the inputs must be changed from the default type of bit to something such as Int or Byte using the Edit Attributes Dialog. After the correct DataSets have been specified for the inputs, connect the inputs and outputs to the add module, and press the Run/Stop Button from the ToolBar ▶.

Behavior Pages

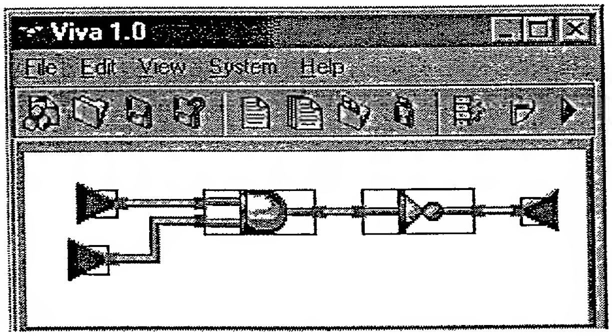
The Behavior Page is located in the left pane of the Graphical User Interface. The Behavior Page is used to define the behavior of a VIVA Module. Unless the Module is Primitive, the Behavior of a Module can usually be displayed by drilling into the object. To drill, either double click on the object or select the object with a left mouse click then click on the descend icon .

You execute a Behavior Page by clicking on the Run/Stop button on the ToolBar.

Definition of VIVA Modules begins with Inputs and Outputs. You place an Input on the Behavior Page by dragging an Input object from the Object Tree, and dropping it onto the Behavior Page, using Drag and Drop. Likewise, you place an Output on the Behavior Page by dragging an Output object from the Object Tree and dropping it onto the Behavior Page, using Drag and Drop.

Information flow should be designed from left to right. Place the Inputs on the left side of the page and the Outputs on the right.

You then define the behavioral relationship between the inputs and the outputs by connecting them, perhaps in series with other Modules, using Transports. The following is the Behavior Page for a NOT, AND (or NAND) Module:

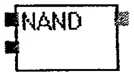


You make the connections or transports between the inputs, outputs, and intervening Modules by clicking on one node, moving the mouse to the connecting node, and then clicking again. (See Connecting Transports). If you have a preferred path, you can click the mouse at various points along the desired path.

Junctions are used when it is necessary to split a signal.

Once the behavior is defined, you then convert the behavior sheet to a project Module. Select the Convert Sheet to Object command from the File Menu, or select the ToolBar icon, .

The system will then create a new Module, in this case called NAND, and place it in the Object Tree. The NAND Module will have one input node and one output node (corresponding to the above Inputs and Output) and appear as follows:

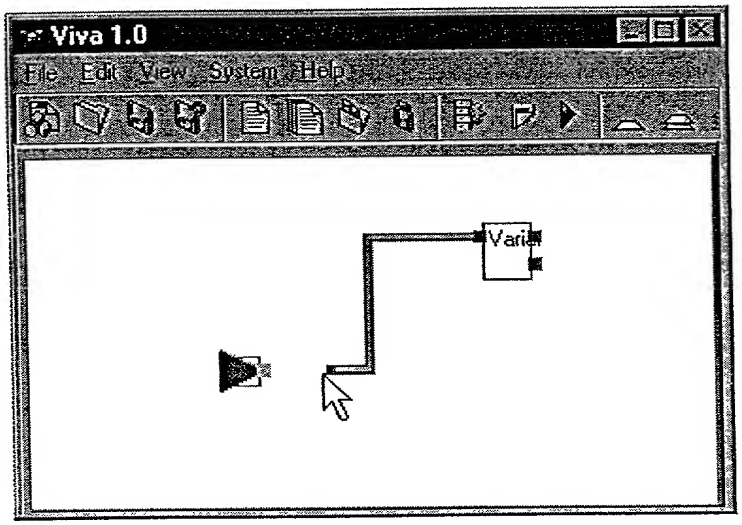


This Module may now be used to construct other behavior pages for Modules with more complex behavior.

Node labels for the inputs and outputs of the NAND Module are the same labels on the inputs and outputs of the Behavior Page.

Connecting Transports

To make the connections or transports between the inputs, outputs, and intervening Modules, click on one node, move the mouse to the connecting node, then click again. If you have a preferred path, you can click the mouse at various points along the desired path.



Notes: To facilitate the correct specification of data types, the following conventions or *rules* have been established:

1. Default data types for Inputs and Outputs are bits. To change the default type, you must either change the DataSet using the Attribute Editor, or connect the Inputs or Outputs to a node with a known type.

2. If a Transport is drawn to an Input or Output with type Bit, this will effect a change of the Input or Output to the DataSet to match that of the node to which the transport is connected.

3. If a Transport is drawn to an arbitrary node of type *Variant*, this will effect a change of the Node DataSet to match that of the node to which the transport is connected.

4. Nodes are either Input Nodes (Source) or Output Nodes (Sink). In general, the Input Nodes are located on the left side of a Module, with the Output Nodes on the right side. Information flow is generally from the left to the right. In general, a Source Node cannot be connected to a Source Node. A transportation path can, however, have multiple Sink Nodes by splitting the signal using Junctions.

5. VIVA will not allow you to connect two nodes with incompatible type.

6. In some instances, the same signal will go to more than one destination. It becomes necessary to create a Junction. This is done by either double clicking on a transport, or by selecting a Junction from the Object Tree.

Hints:

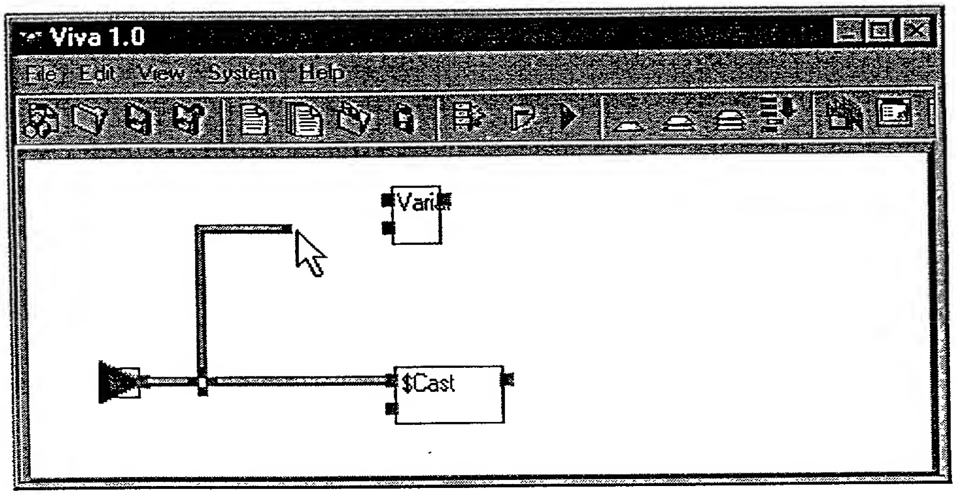
1. Node types become evident by their color if the Show Node Type View Node Type Mode is activated. Do this by clicking on the ToolBar icon .

2. A typical way to force a data type is to connect it to a DataSet Exposer or Collector. For example,

drawing a transport from a Variant Exposer to an Input or an Output will force the DataSet of the Input or Output to be Variant. Moreover, drawing a transport from a Word exposer to an Input will force the DataSet of the input to be of type Word.

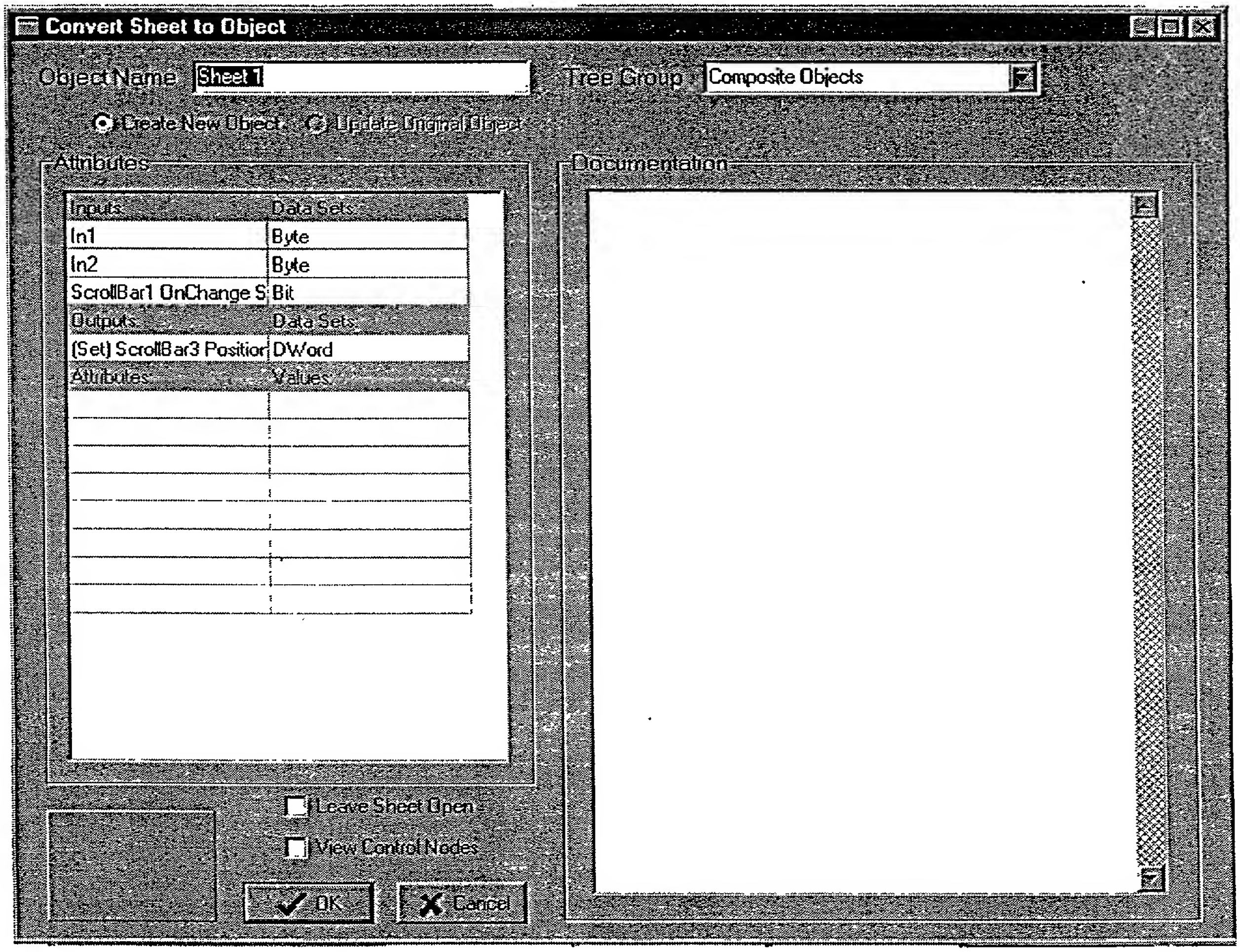
Connecting Junctions

Junctions are used when the same signal needs go to more than one destination. Create a Junction by either double clicking on a transport, or by selecting a Junction from the Object Tree.



Convert Sheet To Object Command

Once you have defined a Behavior Page for a VIVA Module, you can then convert the Behavior Page into a VIVA Module. Inputs and outputs of the Module are the same order and type as defined for the Behavior Page. Before saving the Module, you are given a final chance to enter the information about the Module (things like giving the Module a name, naming the input and output nodes, and documenting the function of the Module and how it is intended to be used). This is done through the Convert Sheet Dialog Box. To maintain universality, this interface is essentially the same as the Edit Attributes Dialog Box.



Convert Sheet to Object

Object Name: Sheet 1

Tree Group: Composite Objects

● Create New Object ○ Update Original Object

Attributes

Inputs	Data Sets
In1	Byte
In2	Byte
ScrollBar1 OnChange S	Bit
Outputs	**Data Sets**
(Set) ScrollBar3 Positior	DWord
Attributes	**Values**

Documentation

□ Leave Sheet Open

□ View Control Nodes

Notes:

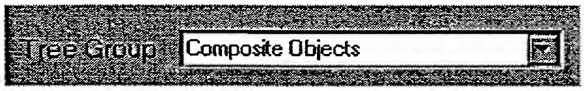
1. This dialog is probably the best interface provided for naming Module nodes. We recommend you name Module nodes here.

2. Documentation should be provided at this point, to define the function of the Module, its intended uses, and restrictions in its use.

3. Overloading is a characteristic that allows you to write several versions of the same behavior to handle the various implementation requirements. As a result, you can have three or four different addition Modules. In many of these cases, you will use a prior implementation as a template for the new

behavior. In these instances, you have the option of either updating the existing Module or creating a new one.

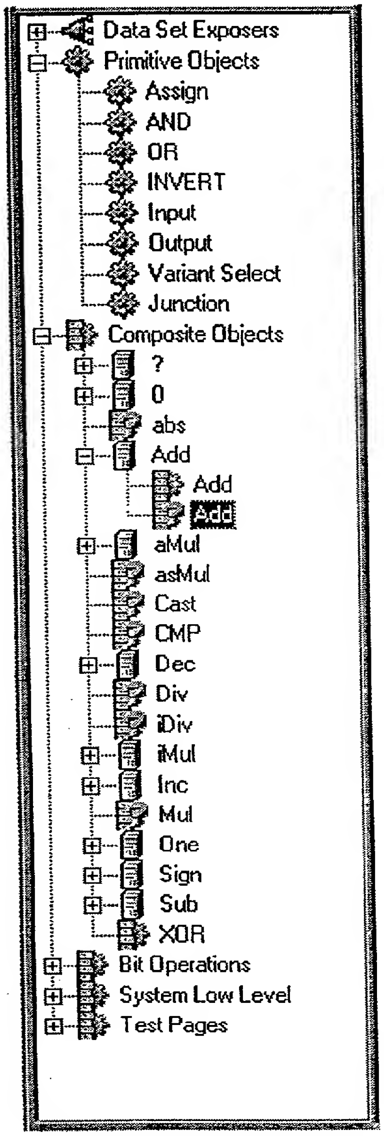
4. You also have the option of selecting for the Module the appropriate Tree Group in the Object Tree. Typical Tree Groups include Bit Operations, Arithmetic Operations, Test Modules, and so forth.



Tree Group: Composite Objects

Object Trees

The Object Tree is usually displayed in the lower right pane of the main VIVA window of the VIVA Graphical User Interface. Object Tree Groups contain predefined Objects or Modules, as well as User-Defined Modules. VIVA also includes fundamental tree groups containing primitive Modules (Bit AND, OR, Invert), DataSet Exposers/Collectors .



The Behavior Page of an individual Module can usually be obtained by double clicking on it. If the Module is needed as part of a VIVA program, it can be placed on the Behavior Page by selecting it, and then dragging it to the appropriate position on the Behavior Page.

Notes:

1. New Tree Groups can be created with a right mouse click in the Object Tree Window.

2. An object can be moved to a different Tree Group through Drag and Drop, or by right clicking on the item, and selecting a different tree.

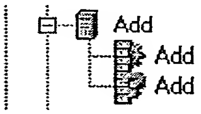
When you create these modules, they are grouped together into one Composite Objectin the Object Tree.

Overloading allows two different Modules to have the same name, but behave differently, or to operate on different DataSets. For example, you can define a Convert module that converts cardinal (unsigned integer) to (signed) integer. The conversion from fixed-point arithmetic to signed integer is a different species. In particular, bits are discarded. In this case, a different behavior is employed.

Composite Objects consist of two or more modules with the same name. When you create these Modules, they are grouped together into one Composite Object. When doing the component layout, you do not need to find the particular Module with the necessary run-time behavior. The composite object can be selected, and the particular Module will be elaborated at compile time. In the above example, the Add, Subtract, Mul, and Divide are all examples of composite objects.

Inputs and Output DataSets of the composite object will generally be Variant unless all of the different implementations agree on the DataSet for a particular node. For example, the Carry node in an Add Module is always of type Bit.

The composing Modules can be obtained by clicking on the Composite Object. The view then expands, revealing all of the instances of the composing Modules. The following is an expansion of the composite Add Object.



The composite Add Object is resolved to two cases. The first case is a Bitwise Add. In some instances, you may intend to do a Bitwise Add, and selecting this Module is appropriate. In other instances, the polymorphic module will handle them.

The issues with creating a general polymorphic Module are covered in Constructing Polymorphic Objects.

Constructing Polymorphic Objects

Polymorphism is the characteristic that allows the same behavioral definition to apply to a large variety of DataSets. Polymorphismis achieved through a combination of Overloading, Recursion, and Run-Time Resolution.

Overloading allows two different Modules to have the same name, but behave differently, or to operate on different DataSets. For example, you can define a Convert module that converts cardinal (unsigned integer) to (signed) integer. The conversion from fixed-point arithmetic to signed integer is a different type of overloading. In particular, Bits are discarded and a different behavior is employed.

When you create these Modules, they are grouped together into one Composite Object in the Object Tree . VIVA resolves the appropriate Module at compile time. In this way, you can program your software to work in a great variety of applications without having to change the Behavior Page.

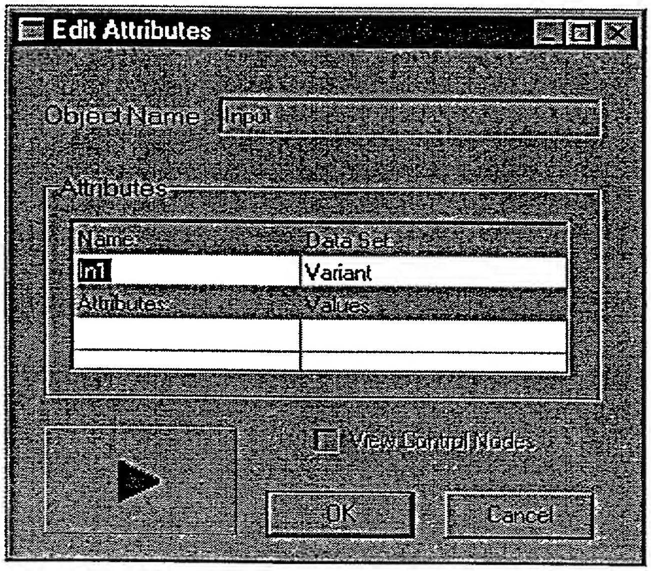
You may ask, "How can one single behavior definition define how to do addition for a variety of 8-bit integers, 16-bit integers, signed and unsigned integers, and fixed-point arithmetic?" The answer is in Recursion.

A graphical object, depicted with input and output nodes, is associated with a series of overloaded functions, each with the same name, which handle the different implementations of a particular function for different types of input. The definitions of Variant Composite Objects can be self-referential, as long as the referenced Modules operate on simpler types than the original types. This rule allows recursive definitions until the function is broken down into an operation on more fundamental data types.

Polymorphism is the ability of VIVA Modules to support data types with arbitrary precision using the same object or function. DataSet Polymorphism refers to VIVA's ability to elaborate the appropriate GateWare implementation depending upon the data type of a variant object's inputs. With Information Rate Polymorphism, VIVA selects a particular implementation depending upon the need for speed.

Overloading is a process whereby multiple objects with the same name can be used to handle different application cases. This Polymorphic property is particularly useful for defining objects to handle Variant situations. Additionally, two different definitions can be specified for the same object.

Modifying an Input



Object Name: Input

Attributes

Name	Data Set
In1	Variant
Attributes	Values

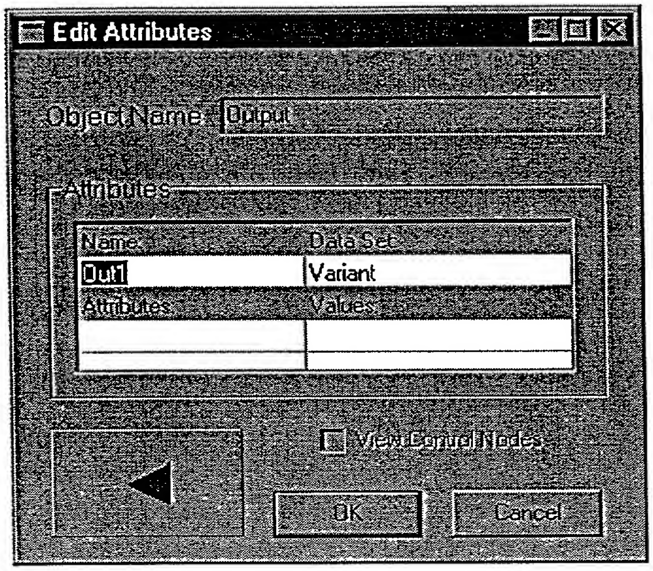
You create inputs to VIVA Modules by dragging an Input object from the Object Tree, then dropping it onto the Behavior Page. To modify the attributes of an Input, right click on the Input object and then enter values in the Edit Attributes Dialog

Data provided to the Input is provided from an external interface. The data will be obtained from the node at the higher-level connection, or from a Widget.

To label an input, enter the appropriate name under the name column. The default name is In1.

The DataSet of the output can be modified by editing the field of the output DataSet. If the desired DataSet is Int, then the string "Int" should appear in the output DataSet. The default DataSet is Bit.

Modifying an Output



Object Name: Output

Attributes

Name	Data Set
Out1	Variant
Attributes	Values

□ View Control Nodes

You create Inputs to VIVA Modules by dragging an Input object from the Object Tree and dropping it onto the Behavior Page. You modify the attributes of an input by right clicking on the Input object, then entering values in the Edit Attributes Dialog.

The output data from the Module will be sent to either a higher-level connection or a Widget.

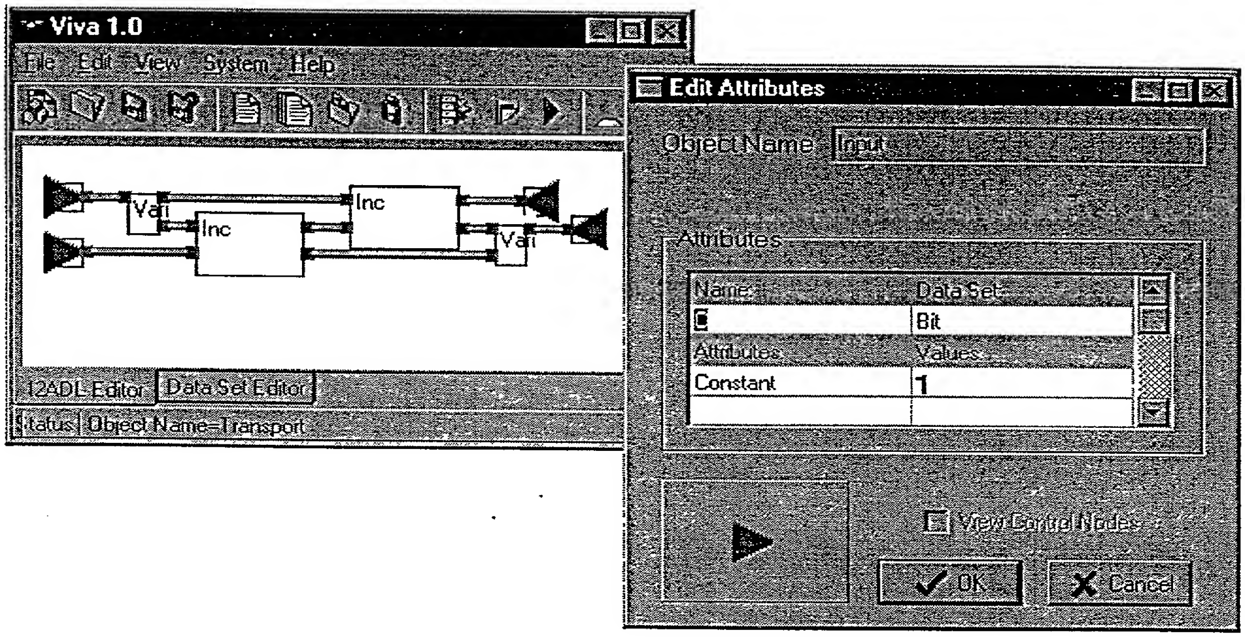
To labeling an output, enter the appropriate name under the name column in the Edit Attributes Dialog Box. The default name is Out1.

The DataSet of the input can be modified by editing the field of the output DataSet. If the desired DataSet is Int, then the string "Int" should appear in the output DataSet. The default DataSet is Bit.

VIVA Constants

VIVA constants are a special case of Inputs. Under ordinary circumstances, the Inputs on a Behavior Page become the nodes of the module at the higher level. By assigning a constant to an Input, this gives the capability to assign values to nodes which are not hooked up.

A good example of this is in the *increment* module which increments an arbitrary number by the value of a bit. In this example the input carry bit is given a default value of 1. This is done through the Edit Attributes Dialog, which is obtained by clicking on the lower input node using the right mouse button. The user then enters *Constant 1* in the attributes section of the dialog box.



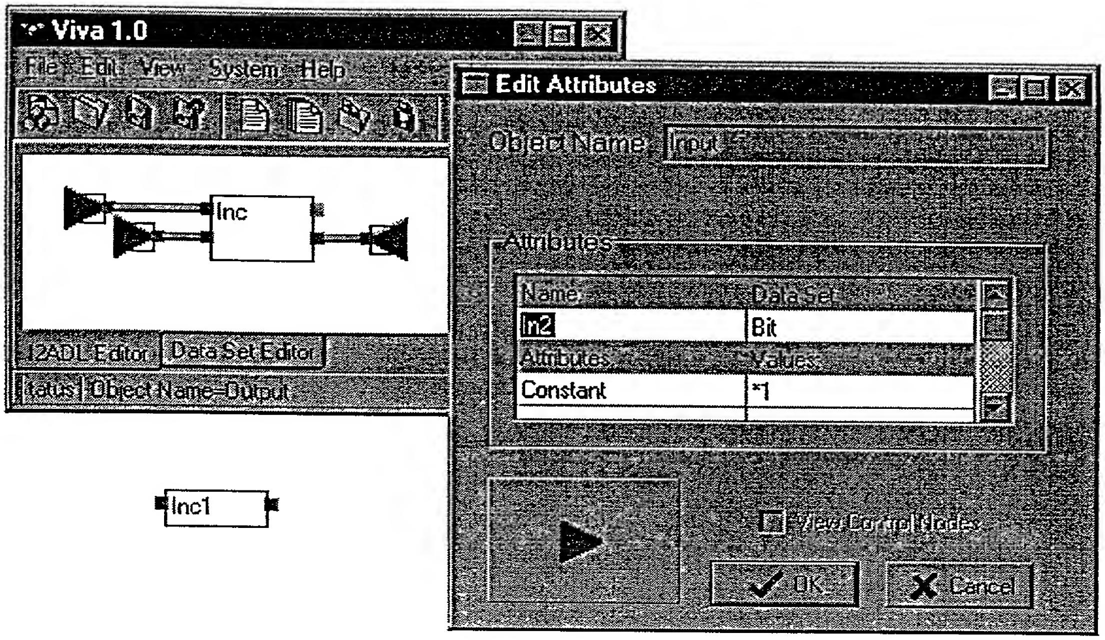
Status: Object Name=Transport

Object Name: Input

Name	Data Set
	Bit
Attributes	**Values**
Constant	1

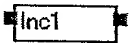
In some applications, the user does not want the constant to appear as an input on the higher-level page. To prevent the constant from being promoted to an input, the user puts an asterisk in front of the constant value.

In the following example, the behavior page for a module, Inc1 increments by 1, but the constant does not become an input node.



Object Name: Input

Name	Data Set
In2	Bit
Attributes	Values
Constant	*1



VIVA DataSets

The fundamental DataSet is the Bit. A Bit is a single piece of information (ON or OFF, 0 or 1, TRUE or FALSE). Other DataSets are formed as aggregates of either bits or more simple DataSets, which are ultimately composed of bits. You can define new DataSets by using the DataSet Editor. In addition to the Bit, VIVA also supports the following fundamental DataSets:

DataSet	Definition
Bit	The atomic DataSet, consisting of a piece of information with two states (TRUE or FALSE, 0 or 1, etc.)
DBit	Double Bit. 2-Bit Cardinal DataSet. Pair of Bits. Half a Nibble. Needed to make higher-level aggregate DataSets.
Nibble	4-Bit Cardinal DataSet. Half a Byte. Needed to make higher level aggregate DataSets.
Byte	8-Bit Cardinal DataSet. Two Nibbles.
Word	16-Bit Cardinal DataSet. Two Bytes.
DWord	Double Word. 32-Bit Cardinal DataSet. Two Words.
QWord	Quad Word. 64-Bit Cardinal DataSet. Four Words.
Int	16-Bit Signed Integer.
DInt	32-Bit Signed Integer.
QInt	64-Bit Signed Integer.
Variant	The data type does not get elaborated until execution, at which time it is resolved into a known DataSet.

A built-in feature of VIVA is the automatic formulation of DataSet Exposers and Collectors. An Exposer Module breaks up a data set into its composing parts; for example, a Word is a concatenation of two Bytes. A Collector combines two parts to make a larger DataSet. For example, the Word Collector combines two Bytes to make a Word.

The Variant DataSet is used to represent data when the actual DataSet is unknown. This important feature enables VIVA to formulate generalized polymorphic Modules that will work independent of the input data types. The VariantOut will compose two arbitrary DataSets into a larger data set. The resolution of a Variant DataSet exposer into the appropriate DataSet exposer will be performed when the input data sets become known.

DataSet Exposers and Collectors

For any VIVA DataSet, the Exposer and Collector are automatically defined as part of the language. An Exposer Module breaks up a DataSet into its component parts; for example, a Word is a concatenation of two Bytes. A Collector combines two parts to make a larger DataSet. For example, the Word Collector combines two Bytes to make a Word. The names of the Exposer and Collector are formulated by appending the characters "In" and "Out" to the DataSet name. For example, the WordIn Exposer breaks up a word into its two composing types. The WordOut Collector takes two Bytes and puts them together to make a word.

DataSet	Exposer	Collector
Bit (Note 1)	-	-
DBit	DBitIn	DBitOut
Nibble	NibbleIn	NibbleOut
Byte	ByteIn	ByteOut
Word	WordIn	WordOut
DWord	DWordIn	DWordOut
QWord	QWordIn	QWordOut
Int	IntIn	IntOut
DInt	DIntIn	DIntOut
QInt	QIntIn	QIntOut
Variant (Note 2)	VariantIn	VariantOut

Note 1: The Bit is the fundamental data set, and has neither a collector nor an exposer. Any attempt to break up a Bit into its composing parts will result in an error.

Note 2: The VariantIn will split an arbitrary DataSet set into two composing parts. This important feature enables VIVA to formulate generalized polymorphic Modules that will work independent of the input data types. The VariantOut will compose two arbitrary DataSets into a larger data set. The resolution of a Variant DataSet exposer into the appropriate DataSet exposer will be performed when the input data sets become known.

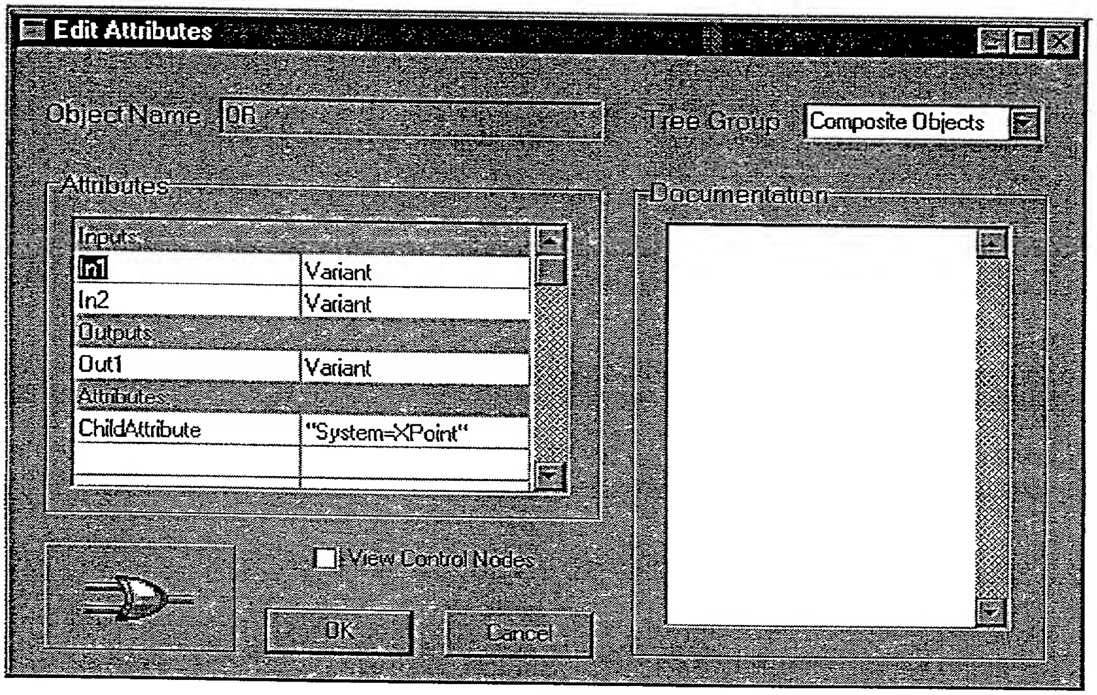
WARNING

In some cases, DataSet Collectors may not be unique. For example a 16-bit signed integer and a 16-bit cardinal (unsigned integer) are both composed from a pair of Bytes. The collector is ambiguous. In this case, VIVA will select the more fundamental exposer. To compose two Bytes into an Int requires that either that IntOut be explicitly specified, or that the output of the Variant Collector be Cast as an integer using the generic system Cast Operator.

Forcing GateWare Allocation

To force any particular VIVA Module into a particular system, set the System Attribute String of the Module to contain the desired system:

1. Right click on the Module. This brings up the Edit Attributes Dialog Box.
2. Under the left attributes column, enter the text, ***ChildAttribute***
3. Under the right attributes column enter the text, ***"System=XPoint"***. (See panel below.)
4. Press the OK button.



Edit Attributes

Object Name: OR

Tree Group: Composite Objects

Attributes

Inputs:	
In1	Variant
In2	Variant
Outputs:	
Out1	Variant
Attributes:	
ChildAttribute	"System=XPoint"

Documentation

□ View Control Nodes

List PopUps

ASIC
Clock
Data-flow machine
Event
Go
Module
Persistence
Reset
VIVA
Behavior
Collector
DBit
Exposer
GUI
Object
Polymorphism
Resolution
Wait
Byte
Composite Object
Done
FPGA
Integer
Overloading
Ready
Sync
Wire list
Cardinal
CPU
Drill
GateWare
Library
Parallel
Recursion
Variant

ASIC
Application-specific integrated circuit

Behavior
The functional characteristics of a VIVA Module. Example: The Behavior of an add object is to take two inputs, add them together, and send the result to the output.

Byte
An aggregate data type consisting of 8 bits. Byte arithmetic is considered to be unsigned integer.

Cardinal
A numeric interpretation where a string of bits is interpreted as an unsigned integer. Byte, Word, DWord, and Huge are all Cardinal number types.

Clock
The VIVA event that is clock driven, usually at regularly timed intervals.

Collector
A Module that assembles a collection of DataSets into a Composite DataSet.

Composite Object
A VIVA Object composed of inputs, outputs, and more primitive Objects connected by Transports.

CPU
Central processing unit for a serial computer, usually a microprocessor.

Data-flow machine
A computer whose data-driven architecture has instructions available for concurrent execution rather than sequential execution.

DBit
Double Bit. 2-Bit Cardinal DataSet. A pair of Bits. Half a Nibble.

> **DBitIn**
> The DataSet Exposer Module of DBit.
>
> **DBitOut**
> The DataSet Collector Module of DBit.

Done
The VIVA event that indicates that the Module has been executed.

Drill
To highlight and to expose additional information about the Module.

Event
In VIVA, a change of state that occurs in time. Examples include ready, wait, go, and done.

Exposer
A VIVA Module that separates a Composite Object into its component parts.

FPGA
Field programmable gate array.

GateWare
A VIVA implementation created through VIVA. GateWare™ is a trademark of Star Bridge Systems, Inc.

Go
The VIVA event that signals another Module to begin execution.

GUI
Graphical user interface.

Integer
A numeric interpretation of a string of bits in which the highest bit is interpreted as a sign bit. Negative values are represented as 2's complement.

Library
A collection of software Objects or Modules that can be accessed by a user and incorporated into a project. VIVA implements Libraries as sheets of objects.

Module
A graphical device with an associated behavior when connected to inputs and/or outputs. Also called Object.

Object
A graphical device with an associated behavior when connected to inputs and/or outputs. For precision, we prefer to use the term Module in VIVA.

Overloading
A process whereby multiple objects with the same name can be used to handle different application cases. This polymorphic property is particularly useful for defining objects to handle variant situations. Additionally, two different definitions can be specified for the same object.

Parallel
The simultaneous execution of multiple circuits rather than a single path.

Persistence
The ability to save VIVA object code (GateWare) on storage media such as a hard disk.

Polymorphism
The ability of a VIVA Module to support data types with arbitrary precision using the same object or function.

> **DataSet Polymorphism** refers to VIVA's ability to instantiate the appropriate GateWare depending upon the data type of a variant object's inputs.
>
> **Information Rate Polymorphism** refers to VIVA's ability to instantiate the appropriate GateWare depending upon the need for speed.

Ready
The VIVA event that notifies another Module that it is ready to receive data.

Recursion
The ability of a function to call itself.

Reset
The VIVA event that signals a Module to reset.

Resolution
The process of determining the meaning of an abstract concept for a particular application.

Sync
The VIVA event that synchronizes the start of execution of two or more processes.

Variant
An attribute that could have multiple meanings, depending on the application.

VIVA
The operating system, language, and user interface created by Kent L. Gilson. VIVA™ is a trademark of Star Bridge Systems, Inc.

Wait
The VIVA event that signals another Module that it is not ready to receive data.

Wire list
The topology of a circuit, showing the devices and their connections. On an FPGA, it shows the state of each gate and the connections between them.

United States Patent & Trademark Office

Office of Initial Patent Examination -- Scanning Division

SCANNED, # 3

Application deficiencies found during scanning:

☒ Page(s)__________ of Specification were not present for scanning.
(Document title)

□ Page(s)__________ of ____________________ were not present for scanning.
(Document title)

□ ***Scanned copy is best available.***